

PATENT  
450117-4840

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

TITLE: METHOD AND SYSTEM FOR COMMUNICATION BETWEEN  
APPLICATION PROGRAMS AND A NETWORK

INVENTORS: Yoeri APTS, Philip MARIVOET

William S. Frommer  
Registration No. 25,506  
FROMMER LAWRENCE & HAUG LLP  
745 Fifth Avenue  
New York, New York 10151  
Tel. (212) 588-0800

655020-1294260

Known network protocols consist of intermediate structure software of layers that are used one on top of the other. The combination of these layers or intermediate structure software is referred to as network protocol stack. At the bottom of these stacks the physical drivers are arranged, which drivers are responsible for sending data to and receiving data from the actual network. At the top of the stack the user application program is arranged, which program generates and consumes data. Data used by a user application program travels, before being sent onto the physical network, down through the protocol stack, where it is manipulated by every protocol layer before finally arriving at the bottom layer where the manipulated data is put onto the physical network. Similarly, when data is received by the physical drivers, data travels upward through the protocol stack before it is delivered to the user application.

**THE UNIVERSITY OF CHICAGO**

- THE UNIVERSITY OF CHICAGO**

**THE UNIVERSITY OF CHICAGO**

**THE UNIVERSITY OF CHICAGO**

- THE UNIVERSITY OF CHICAGO**

5 internal events.

the network protocol stack.

10 modules is accomplished through interconnecting queue-

15 than one queue can be implemented, wherein different

25 "encapsulation". When a data unit moves up the stack,  
this protocol control information is stripped off again.

35 to the rules of communication with the network without

rebooting the computer system.

- figure 1 is a diagram shows schematically a personal computer connected to a network;

- figure 2 is a diagram showing a known layer protocol stack;

- figure 3 is a diagram illustrating a method and system according to a preferred embodiment of the method and system according to the present invention;

- figure 4 is diagram showing a memory pool shared between program objects;

- figure 5a is a diagram showing the memory layout of a data unit;

10 - figure 5b is a diagram showing the logical structure of the data unit;

- figure 6a is a diagram showing the memory layout of a data unit to which a UDP header is added;

- figure 6b is a diagram showing a logical structure the data unit to which the UDP header is added;

- figure 7a is a diagram showing a memory layout of figure 6a after fragmentation in two IP packets and after insertion of headers; and

20 - figure 7b is a diagram showing the logical structure of figure 6b after fragmentation in two IP packets and after insertion of headers.

Figure 1 shows a personal computer or workstation that comprises a central processing unit 1, a random access memory 2, a read only memory 3 and a network adapter 5, all of which are connected through a bus 4. The network adapter 5 is attached to a network 7 which is in turn connected to a host computer 6. Instead of a personal computer a wide and varied range of devices can be used ranging from small embedded systems like cellular phones, to large high-performance video servers.

Network protocols are known that are combined into a layered structure, for example the network protocols of the OSI model, which is a standard for worldwide communications defining a framework for implementing protocols in seven layers.

In figure 2 a four layer protocol stack is shown, in which application program layer 8 handles the details of a particular application program, transport

0044571-00000

layer 9 provides a flow of data between two hosts, the network layer 10 handles the movement of data units around the network and the link layer 11 includes the network device driver program handling the actual physical interfacing with the network.

The internet model (TCP/IP) is a four layer protocol stack. Layer 8 includes a file transfer protocol (FTP) for downloading or uploading files, a simple mail transfer protocol (SMTP) for electronic mail, etc.. The transport layer 9 is the Transmission Control Protocol (TCP) that is responsible for verifying the correct delivery of data. TCP adds support information to detect errors or lost data and to trigger retransmission until the data is correctly and completely received. Another transport layer 9 is the User Datagram Protocol (UDP), that sends data units (datagrams) from one host to another, without checking whether the data units reach their destination. In this embodiment the application program is responsible for reliable delivery thereof. As an example of a network layer 10 the Internet Protocol (IP) provides a routing mechanism that routes a message across the network. Application programs such as the File Transfer Protocol (FTP) or the Simple Mail Transfer Protocol (SMTP) are provided to respectively allow for downloading and uploading between different network sites and to allow for sending and receiving electronic mail messages from and to the network sites. A link layer 11 is provided for implementing ATM, IEEE 1394 or ethernet networks.

According to the preferred embodiment a specialized execution environment or also called metaspace is provided for the program objects which run on top of it. Each metaspace has for example its own message passing scheme that is adapted to the specific needs thereof. The specialized metaspace for network protocols and network protocol stacks is called mNet, the metaspace that supports device driver programs is called

mDrive, and the metaspace for programming an application program is called mCOOP.

When a data unit is received by the physical drivers on mDrive, it travels through the protocol objects on mNet before it is consumed by the user application running on mCoop. Summarizing, the device driver programs run on mDrive, the network protocols objects run on mNet and application objects run on mCoop. The advantage of building these dedicated environments, for example the mNet for the implementation of protocol and protocol stacks, is that maximum support for the specific functionalities needed is provided.

In the mNet metaspace, a network protocol is implemented through one or more program objects called modules with a predefined set of methods or function. A protocol stack is implemented by interconnecting multiple modules with queue objects. Data units travel through a set of interconnecting modules and each module performs some actions on the data units, that is, it adds or removes headers and trailers, fragments or defragments the data units etc. before passing it onto the next module.

In figure 3 the mNet metaspace and its position amongst the other metaspaces is shown. When a data unit is generated by an application program object 10 or 11 on mCoop, this data unit travels down the protocol objects 12-16 running on mNet, where it is manipulated by every program object. Program object 12 is in the internet model the Transmission Control Protocol (TCP), program object 13 is the User Datagram Protocol (UDP), program object 14 is the Internet Protocol IP, program object 15 is the Point to Point Protocol, which provides dial-up access over serial communication lines, and program object 16 provides an ATM-interface.

When the manipulated data unit arrives at the bottom layer, it is put on a physical network by the device driver programs 17 or 18 running on mDrive which provide a serial driver and an ATM-driver respectively.

0044271-00099



5

hereafter:

- 10

- 15

- 20

- 25

30

35

queue with a size that differs from that of the size of the SDUs that were originally written by the sender. Also data read from a streaming queue must be explicitly acknowledged before it is actually removed from the  
5 queue.

In another preferred embodiment program objects or modules can be interconnected using more than one queue. Using a priority mechanism, one queue can be used for normal traffic, while another queue can be used for  
10 expedited data transfer. The data from the expedited queue will be offered first to the module by the system.

In another preferred embodiment of the invention the program objects are interconnected bidirectionally by interconnecting queues, which queues  
15 have two priority levels for passing SDUs, i.e. normal priority and expedited priority. When an SDU is sent on a queue with expedited priority, it will arrive at the other end of the queue before all other SDU with normal priority.

20 The basic data unit that is used to pass information between the modules is called the Service Data Unit (SDU). SDUs are dynamic memory buffers, shared by all modules, used for data manipulations whereby physical copying of data is avoided as much as possible.

25 A module always iterates through the following steps:

- receive a SDU from a queue;
- process the SDU and update the internal state of the module;
- 30 - send one or more SDUs to a next module;
- optionally perform postprocessing; and
- wait for the next SDU to be received.

Modules can be interconnected using queues at  
35 run time. These queues can also be removed at any time. This allows for the dynamic creation and configuration of protocol stacks.

ORIGINAL - 020889

A coding example showing in what way a protocol stack on mNet can be dynamically built, is given below.

The code first looks for all protocol stack modules using the mNet::Find service. The modules are then

5 interconnected with atomic queues using the mNetQueue::Open service.

```

void mNetMain() {
    SID sidLoop;
10    SID sidATMiface;
    --SID sidIP;
    SID sidUDP;
    SID sidTCP;

15    InterfaceInfo interface;
    ProtocolInfo protocol;

    do {
        cout << "= INITIALISING STACK =" << endl;
20        cout << "Locating stack components" << endl;
        mNet::Find("LoopInterface",sidLoop);
        mNet::Find("ATMiface",sidATMiface);
        mNet::Find("IP",sidIP)
        mNet::Find("UDP",sidUDP);
25        mNet::Find("TCP",sidTCP);

        // IP and Loopback HAVE to be in the image.
        if (!(sidIP.IsValid() && sidLoop.IsValid())){
            cout << RED "Can't build stack..." << endl;
30            break;
        }
        cout << "Connecting components" << endl;

        interface.address = IPAddress("127.0.0.0");
35        interface.netmask = IPAddress("255.0.0.0");
        interface.flags = IFF_LOOPBACK;
    }
}

```

```

mNetQueue::Open(sidIP, BOTTOM,
                sidLoop, TOP,
                FLOW_ATOMIC, 0,
                sizeof(InterfaceInfo),
5                &interface
                );

if (sidATMiface.IsValid()) {
    interface.address =
10    IPAdress((char*)myipaddr);
    interface.netmask =
        IPAdress("255.255.255.0");
    interface.flags = 0x00;
    mNetQueue::Open(sidIP, BOTTOM,
15        sidATMiface, TOP,
        FLOW_ATOMIC, 0,
        sizeof(InterfaceInfo),
        &interface
        );
20 }

if (sidUDP.IsValid()) {
    protocol.protocol = IPPROTO_UDP;
    mNetQueue::Open(sidUDP, BOTTOM,
25        sidIP, TOP,
        FLOW_ATOMIC, 0,
        sizeof(ProtocolInfo),
        &protocol
    );
30 }

if (sidTCP.IsValid()) {
    protocol.protocol = IPPROTO_TCP
    mNetQueue::Open(sidTCP, BOTTOM,
35        sidIP, TOP,
        FLOW_ATOMIC, 0,
        sizeof(ProtocolInfo),
        &protocol

```

00000000-00000000

```

        );
    }
    cout << "= DONE =" << endl;
} while(0);
5 }

```

Data units travel through a set of interconnecting modules and each module performs some actions on the data units, before passing them to the next module. Since all these manipulations need to be performed as fast as possible, data units are not necessarily copied. Instead data references pointing to data units and queues are passed between the modules. Therefore all SDUs are managed by a central SDU Manager which manages a memory pool of data units. This memory pool is shared between the mNet SDU manager and all mNet modules and queues. In figure 4 SDUs 23 and 24 of program objects 20 and 21 are in the memory pool 22 which is managed by the SDU Manager 25. To avoid physical copying or moving of SDUs 23 or 24 in the shared memory pool 22 when going from one program object 20 to another program object 21, the SDUs are stored using references, which references point to the memory location of the SDU, and offsets and sizes of the data units within the SDU, as is shown in figures 5-7.

Consider a program application that sends data units over an ATM network using the User Datagram Protocol (UDP). The application program first builds a SDU that contains the application data units. The original SDU memory lay-out of memory pool 22 is shown in figure 5a and the logical structure thereof is shown in figure 5b. The data contained in the SDU is stored in variable sized data units and the actual SDU is constructed by linking data units together at certain offsets. The SDU is then headed over to the UDP module. The UDP module adds the appropriate UDP header in front of the SDU. The effect of this action on the SDU organization is illustrated in figure 6a in which the

10 The only actions involved are reorganizations of  
references to the SDU and reorganizations of offsets and  
sizes of a data unit in the SDU. This makes the  
communication method fast and reliable.

15 the following advantages:

- data can be shared between SDUs, that is copying or moving of data from one SDU to another is done by keeping references and reference counters instead of physically copying or moving the data;

- 20                   - adding data and removing data from an SDU can  
be accomplished without physically copying or moving the  
SDU contents;

- SDU data does not have to be stored contiguously in memory.

25 In a further preferred embodiment the SDUs are organized, i.e. created, allocated, etc., in SDU pools, which are shared memory buffers. Every program object or module has attached to it one single SDU pool in which it creates SDUs and allocates data for the SDUs. In a  
30 preferred embodiment the intermediate structure software and the application program have different SDU pools which are optimized for their specific use. SDU pools are shareable amongst more than one program object or module. An advantage thereof is that important program objects  
35 can be allocated a larger SDU pool while a smaller SDU pool will suffice in case of a less important program objects. The SDU pools also provide indirectly for the local flow control mechanism of the protocol stack.

5 internal communication mechanism of the specific hardware configuration and symbolic names. These names can be mapped to the earlier described references and vice versa.

10 above given embodiments. The scope of the present invention is defined by the next claims, while many modifications to the above embodiments are possible within the scope thereof.

